MULTIMEDIA UNIVERSITY

STUDENT ID NO

MULTIMEDIA UNIVERSITY

FINAL EXAMINATION

TRIMESTER 2, 2015/2016

ECE3166 ADVANCED MICROPROCESSORS

(All Majors)

29 FEBRUARY 2016
2:30 P.M. – 4:30 P.M.
(2 Hours)

INSTRUCTIONS TO STUDENTS

1. This Question paper consists of 13 pages with FOUR (4) Questions only.
2. Attempt **ALL** questions. All questions carry equal marks and the distribution of the marks for each question is given.
3. Please print all your answers in the Answer Booklet provided.
4. The summary of instruction set and the formats of some relevant descriptors and table entry are provided in the appendices.

Question 1

(a) Explain the fetch-execute cycle of a von Neumann machine.

[4 marks]

(b) Draw a table to contrast FOUR (4) aspects of the RISC and CISC architecture. Name one example product for the RISC and CISC microprocessors, respectively.

[5 marks]

(c) Find the memory capacity of each microprocessor in Table Q1 in **bytes**. Rank the microprocessors below in ascending order (small to big) according to their memory capacity.

Table Q1.

Microprocessor	Width of memory block (bits)	Size of address bus (bits)
A	4	16
B	8	20
C	16	12
D	32	10

[5 marks]

(d) The following data is stored in the memory of a 8086 microprocessor starting from address 20200H. Illustrate the data location in the memory. The memory addresses and their corresponding memory content must be specified in **hexadecimal**.

$$305441741_{10}$$

[3 marks]

(e) P6 processors implements the dynamic execution microarchitecture. Sketch a block diagram that describes the three core engine interface with memory via unified caches.

[8 marks]

Continued ...

Question 2

(a) Identify the addressing modes for the destination and source operands for the following real-mode assembly instructions:

	Assembly instruction	Addressing mode	
		Destination	Source
(i)	MOV BL, E4H	?	?
(ii)	MOV [ESI*3]+20H,CX	?	?

[4 marks]

(b) Checkerboard pattern shown in Figure Q2 is used in testing of microchip to detect bit-flip (from '1' to '0' or from'0' to '1') occurance in the memory. 32KB of checkerboard pattern needs to be written starting from address ES:200H.

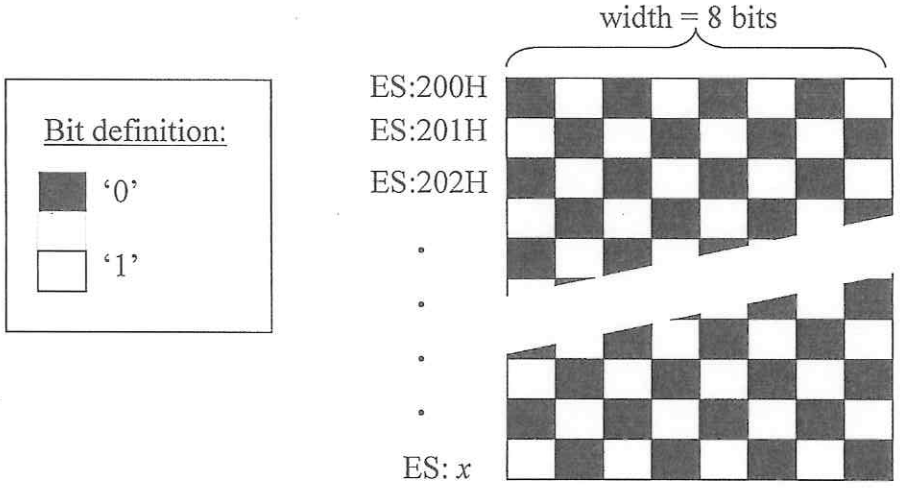


Figure Q2

(i) Specify the ending logical address (ES: x).

[2 marks]

(ii) Modify the code in Listing Q2 to write the checkerboard pattern using the x86 string handling instructions.

```
MOV CX, 1234H
MOV SI, 0100H
MOV DI, 0200H
STD
REPE CMPSW
```

Listing Q2

[7 marks]

Continued ...

Question 2

(c) A fragment of an x86 program is given in Table Q2. Redraw the table in your answer booklet and fill up the empty boxes. State the contents in **hexadecimal**.

Table Q2

Instruction	**Affected register(s) = Content**
MOV BL, 85	BL = 55H
SUB BL, 80	
MOV AL, 2	
IMUL BL	
MOV CX, AX	
MOV AL, -118	
ADD AL, 10	
CBW	
CWD	
IDIV CX	

[12 marks]

Continued ...

Question 3

(a) The protected mode uses memory pointer that are 48-bits in length.

(i) List the two parts of a virtual address.

[1 mark]

(ii) How large is the 80386DX virtual address space and what is the maximum number of segments that can exist?

[3 marks]

(iii) Identify which part of the processor is used to translate virtual address to physical address.

[1 mark]

(b) A code segment selector register is loaded with a value of 2006H. The base address of Global Descriptor Table (GDT) and Local Descriptor Table (LDT) are 0300 0000H and 0800 0000H, respectively.

[03002000H] = 00FFH	[08002000H] = 01FFH
[03002002H] = 0000H	[08002002H] = 0000H
[03002004H] = 9200H	[08002004H] = DF10H
[03002006H] = 0900H	[08002006H] = 0100H

(i) Determine the address range of the code segment descriptor.

[3 marks]

(ii) Find the base address and the limit of the segment.

[3 marks]

(iii) Examine and interpret the access right byte of the segment descriptor.

[3 marks]

(c) An instruction is trying to access an operand which is pointed by a virtual address of 1007H:00024400H.

(i) Evaluate the requested privilege level.

[3 marks]

(ii) Justify whether the segment descriptor is located in GDT or LDT.

[2 marks]

(iii) If the base address of LDT is 0100 0000H and GDT is 0200 0000H, evaluate the starting address and the ending address of the segment descriptor.

[4 marks]

(iv) If the segment base address read from the descriptor is 0300 0000H, compute the physical address of the operand.

[2 marks]

Continued ...

Question 4

(a) Codes in Listing Q4 employ the MMX instructions.

```
struct SVec
{     short a, b, c, d; //Word-sized data
};
struct SVec MMX_Add(struct SVec op_a, struct SVec op_b)
{     struct SVec ret_vector;
      int result = 0;
      __asm
      {     MOVQ     MM0, op_a ; //MM0 = 0028001E0014000A16
            MOVQ     MM1, op_b ; //Question Q4(a)(ii)
            PMADDWD  MM0, MM1  ; //Question Q4(a)(iii)
            MOVQ     ret_vector, MM0; //Save the return vector
      }
      return ret_vector;
};
int main()
{     struct SVec obj1; struct SVec obj2; struct SVec res;

      //0x0A = hexadecimal value A
      obj1.a=0x0A; obj1.b=0x14; obj1.c=0x1E;obj1.d=0x28;
      obj2.a=0x05; obj2.b=0x06; obj2.c=0x07;obj2.d=0x08;

      // Call the function, MMX_Add.
      res = MMX_Add(obj1, obj2);
      printf("MMX add: %i,%i,%i,%i\n",
            res.a, res.b, res.c, res.d); //Question Q4(a)(iv)
      return 0;}
```

Listing Q4

(i) How many bits can be stored in a single MMX register? How many word-sized elements can be stored in the MMX register?

[2 mark]

(ii) What is the content of the destination operand after execution of the following instruction "`MOVQ MM1, op_b`"? Write the answer in **hexadecimal**.

[2 mark]

(iii) What is the content of the destination operand after execution of the following instruction "`PMADDWD MM0, MM1`" ? Write the answer in **hexadecimal**.

[4 marks]

(iv) What is displayed at the output after the `printf` instruction?

[2 marks]

(v) Modify the code in Listing Q4 to perform packed addition of four single precision floating point numbers using SSE instructions. Re-write the entire fragment.

[5 marks]

Continued ...

Question 4

(b) Briefly discuss the disadvantages of 64-bit processor compared to 32-bit processor.

[3 marks]

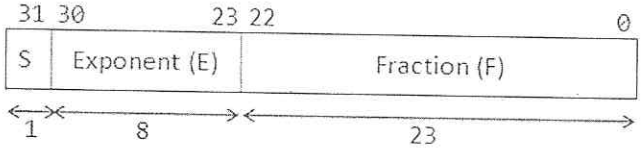
(c) Multicore technology refers to multiple individual processors (cores) on a single die. List the drawbacks/challenges in multicore processor systems.

[4 marks]

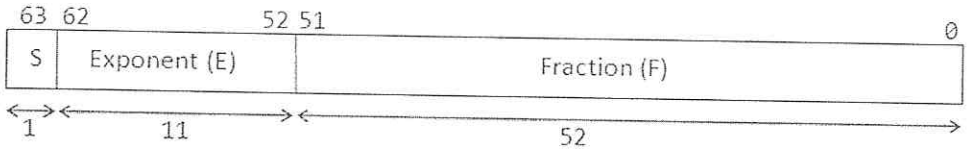
(d) With the aid of a diagram, briefly describe the Intel Hyper-Threading Technology.

[3 marks]

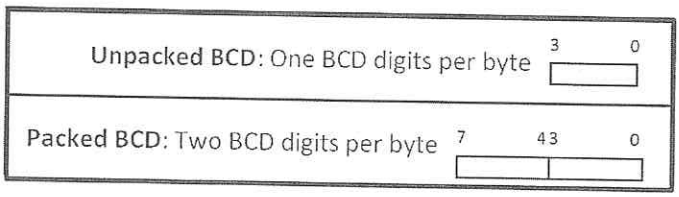
Appendix A



32-bit Single-Precision Floating-point Number



64-bit Double-Precision Floating-point Number



Appendix B

Type of Memory Reference	Default Segment	Alternate Segment
Instruction fetch	Code Segment CS	None
Stack operation	Stack Segment SS	None
General data	Data Segment DS	CS, ES, SS
String source	Data Segment DS	CS, ES, SS
String destination	Extra Segment ES	None
BX used as pointer	Data Segment DS	CS, ES, SS
BP used as pointer	Stack segment SS	CS, ES, DS

Continued ...

Appendix C

Instruction Set Summary for Intel 8086/80186/80286/80386/80486

AAA - Ascii Adjust for Addition
AAS - Ascii Adjust for Subtraction
ADC - Add With Carry
ADD - Arithmetic Addition
AND - Logical And
BSF - Bit Scan Forward (386+)
BSR - Bit Scan Reverse (386+)
BSWAP - Byte Swap (486+)
BT - Bit Test (386+)
BTC - Bit Test with Compliment (386+)
BTR - Bit Test with Reset (386+)
BTS - Bit Test and Set (386+)
CALL - Procedure Call
CBW - Convert Byte to Word
CDQ - Convert Double to Quad (386+)
CLC - Clear Carry
CLD - Clear Direction Flag
CLI - Clear Interrupt Flag (disable)
CLTS - Clear Task Switched Flag (286+ privileged)
CMC - Complement Carry Flag
CMP - Compare
CMPS - Compare String (Byte, Word or Doubleword)
CMPXCHG - Compare and Exchange
CWD - Convert Word to Doubleword
CWDE - Convert Word to Extended Doubleword (386+)
DAA - Decimal Adjust for Addition
DAS - Decimal Adjust for Subtraction
DEC - Decrement
DIV - Divide
ENTER - Make Stack Frame (80188+)
ESC - Escape
HLT - Halt CPU
IDIV - Signed Integer Division
IMUL - Signed Multiply
IN - Input Byte or Word From Port
INC - Increment
INT - Interrupt
INTO - Interrupt on Overflow
INVD - Invalidate Cache (486+)
INVLPG - Invalidate Translation Look-Aside Buffer Entry (486+)
IRET/IRETD - Interrupt Return
Jxx - Jump Instructions Table
JCXZ/JECXZ - Jump if Register (E)CX is Zero
JMP - Unconditional Jump
LAHF - Load Register AH From Flags
LAR - Load Access Rights (286+ protected)
LDS - Load Pointer Using DS
LEA - Load Effective Address
LES - Load Pointer Using ES
LFS - Load Pointer Using FS (386+)
LGDT - Load Global Descriptor Table (286+ privileged)
LIDT - Load Interrupt Descriptor Table (286+ privileged)
LGS - Load Pointer Using GS (386+)
LLDT - Load Local Descriptor Table (286+ privileged)
LMSW - Load Machine Status Word (286+ privileged)

Continued ...

LOCK - Lock Bus
LODS - Load String (Byte, Word or Double)
LOOP - Decrement CX and Loop if CX Not Zero
LOOPE/LOOPZ - Loop While Equal / Loop While Zero
LOOPNZ/LOOPNE - Loop While Not Zero / Loop While Not Equal
LSL - Load Segment Limit (286+ protected)
LSS - Load Pointer Using SS (386+)
LTR - Load Task Register (286+ privileged)
MOV - Move Byte or Word
MOVS - Move String (Byte or Word)
MOVSX - Move with Sign Extend (386+)
MOVZX - Move with Zero Extend (386+)
MUL - Unsigned Multiply
NEG - Two's Complement Negation
NOP - No Operation (90h)
OR - Inclusive Logical OR
OUT - Output Data to Port
POP - Pop Word off Stack
POPA/POPAD - Pop All Registers onto Stack (80188+)
POPF/POPFD - Pop Flags off Stack
PUSH - Push Word onto Stack
PUSHA/PUSHAD - Push All Registers onto Stack (80188+)
PUSHF/PUSHFD - Push Flags onto Stack
RCL - Rotate Through Carry Left
RCR - Rotate Through Carry Right
REP - Repeat String Operation
REPE/REPZ - Repeat Equal / Repeat Zero
REPNE/REPNZ - Repeat Not Equal / Repeat Not Zero
RET/RETF - Return From Procedure
ROL - Rotate Left
ROR - Rotate Right
SAHF - Store AH Register into FLAGS
SAL/SHL - Shift Arithmetic Left / Shift Logical Left
SAR - Shift Arithmetic Right
SBB - Subtract with Borrow/Carry
SCAS - Scan String (Byte, Word or Doubleword)
SETS - Set if Signed (386+)
SETNS - Set if Not Signed (386+)
SETC - Set if Carry (386+)
SETNC - Set if Not Carry (386+)
SETO - Set if Overflow (386+)
SETP/SETPE - Set if Parity / Set if Parity Even (386+)
SETNP/SETPO - Set if No Parity / Set if Parity Odd (386+)
SGDT - Store Global Descriptor Table (286+ privileged)
SIDT - Store Interrupt Descriptor Table (286+ privileged)
SHL - Shift Logical Left
SHR - Shift Logical Right
SHLD/SHRD - Double Precision Shift (386+)
SLDT - Store Local Descriptor Table (286+ privileged)
SMSW - Store Machine Status Word (286+ privileged)
STC - Set Carry
STD - Set Direction Flag
STI - Set Interrupt Flag (Enable Interrupts)
STOS - Store String (Byte, Word or Doubleword)
STR - Store Task Register (286+ privileged)
SUB - Subtract
TEST - Test for Bit Pattern
WBINVD - Write-Back and Invalidate Cache (486+)
XCHG - Exchange
XOR - Exclusive OR

Continued ...

Mnemonic	Meaning	Format	Operation
LOOP	Loop	LOOP Short-label	(CX) ← (CX) − 1 Jump is initiated to location defined by short-label if (CX) ≠ 0; otherwise, execute next sequential instruction
LOOPE/LOOPZ	Loop while equal/ loop while zero	LOOPE/LOOPZ Short-label	(CX) ← (CX) − 1 Jump to location defined by short-label if (CX) ≠ 0 and (ZF) = 1; otherwise, execute next sequential instruction
LOOPNE/ LOOPNZ	Loop while not equal/ loop while not zero	LOOPNE/LOOPNZ Short-label	(CX) ← (CX) − 1 Jump to location defined by short-label if (CX) ≠ 0 and (ZF) = 0; otherwise, execute next sequential instruction

Continued ...

Mnemonic	Meaning	Format	Operation	Flags Affected
MOVS	Move string	MOVSB/MOVSW	((ES)0 + (DI)) ← ((DS)0 + (SI)) (SI) ← (SI) ± 1 or 2 (DI) ← (DI) ± 1 or 2	None
CMPS	Compare string	CMPSB/CMPSW	Set flags as per ((DS)0 + (SI)) − ((ES)0 + (DI)) (SI) ← (SI) ± 1 or 2 (DI) ← (DI) ± 1 or 2	CF, PF, AF, ZF, SF, OF
SCAS	Scan string	SCASB/SCASW	Set flags as per (AL or AX) − ((ES)0 + (DI)) (DI) ← (DI) ± 1 or 2	CF, PF, AF, ZF, SF, OF
LODS	Load string	LODSB/LODSW	(AL or AX) ← ((DS)0 + (SI)) (SI) ← (SI) ± 1 or 2	None
STOS	Store string	STOSB/STOSW	((ES)0 + (DI)) ← (AL or AX) ± 1 or 2 (DI) ← (DI) ± 1 or 2	None

Prefix	Used with:	Meaning
REP	MOVS STOS	Repeat while not end of string CX ≠ 0
REPE/REPZ	CMPS SCAS	Repeat while not end of string and strings are equal CX ≠ 0 and ZF = 1
REPNE/REPNZ	CMPS SCAS	Repeat while not end of string and strings are not equal CX ≠ 0 and ZF = 0

Mnemonic	Meaning	Format	Operation	Flags Affected
CLD	Clear DF	CLD	(DF) ← 0	DF
STD	Set DF	STD	(DF) ← 1	DF

Continued ...

Appendix D

MMX Instruction Set Summary

Mnemonic	Name
MOVQ	Move quad word
MOVD	Move double word
PADDB	Packed add bytes
PADDW	Packed add words
PADDD	Packed add double words
PADDSB	Packed add signed bytes
PADDSW	Packed add signed words
PADDUSB	Packed add unsigned bytes
PADDUSW	Packed add unsigned words
PSUBB	Packed subtract bytes
PSUBW	Packed subtract words
PSUBD	Packed subtract double words
PSUBSB	Packed subtract signed bytes
PSUBSW	Packed subtract signed words
PSUBSD	Packed subtract signed double words
PSUBUSB	Packed subtract unsigned bytes
PSUBUSW	Packed subtract unsigned words
PMULLW	Packed multiply low words
PMULHW	Packed multiply high words
PMADDWD	Packed multiply and add words
PSRAW	Packed shift right arithmetic words
PSRAD	Packed shift right arithmetic double words
PCMPEQB	Packed compare for equal bytes
PCMPEQW	Packed compare for equal words
PCMPEQD	Packed compare for equal double words
PCMPGTB	Packed compare for greater than bytes
PCMPGTW	Packed compare for greater than words
PCMPGTD	Packed compare for greater than double words
PAND	Bitwise logical AND
PANDN	Bitwise logical AND NOT
POR	Bitwise logical OR
PXOR	Bitwise logical exclusive-OR
PSLLW	Packed shift left logical word
PSLLD	Packed shift left logical double word
PSLLQ	Packed shift left logical quad word
PSRLW	Packed shift right logical word
PSRLD	Packed shift right logical double word
PSRLQ	Packed shift right logical quad word
PACKUSWB	Pack words to bytes with unsigned saturation
PACKSSWB	Pack words to bytes with signed saturation
PACKSSDW	Pack double words to words with signed saturation
PUNPCKHBW	Unpack high packed data bytes to words
PUNPCKHWD	Unpack high packed data words to double words
PUNPCKHDQ	Unpack high packed data double words to quad word
PUNPCKLBW	Unpack low packed data bytes to words
PUNPCKLWD	Unpack low packed data words to double words
PUNPCKLDQ	Unpack low packed data double words to quad word

Continued ...

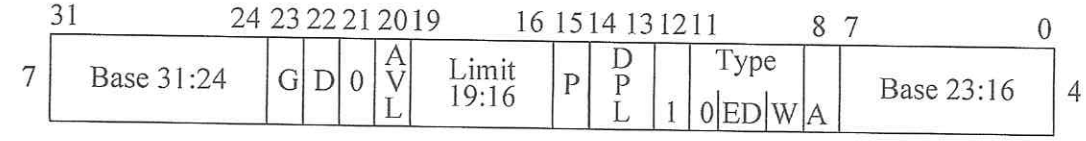
Appendix E

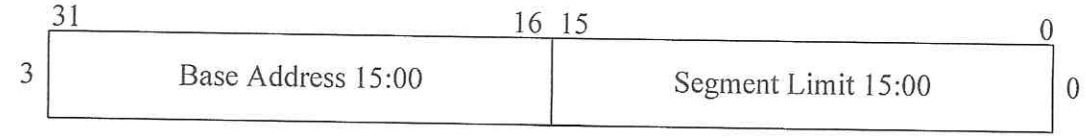
Summary of Selected SSE Instructions

Mnemonic	Description
MOVAPS	Move aligned packed single-precision floating-point values
MOVUPS	Move unaligned packed single-precision, floating-point
MOVHPS	Move high packed single-precision floating-point
MOVHLPS	Move packed single-precision floating-point high to low
MOVLHPS	Move packed single-precision floating-point low to high
MOVLPS	Move low packed single-precision floating-point
MOVMSKPS	Move packed single-precision floating-point mask
MOVSS	Move scalar single-precision floating-point
ADDSS	Scalar add pair of operands
ADDPS	Packed add pairs of operands
SUBSS	Scalar subtract pair of operands
SUBPS	Packed subtract pairs of operands
MULSS	Scalar multiply pair of operands
MULPS	Packed multiply pairs of operands
DIVSS	Scalar divide pair of operands
DIVPS	Packed divide pairs of operands

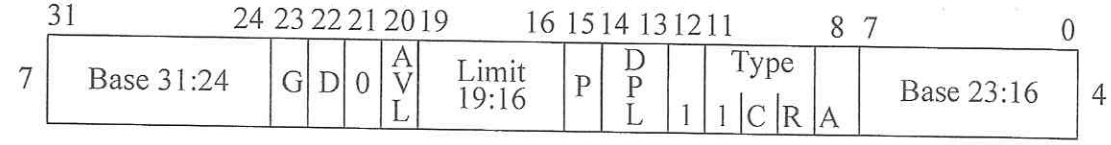
Appendix F

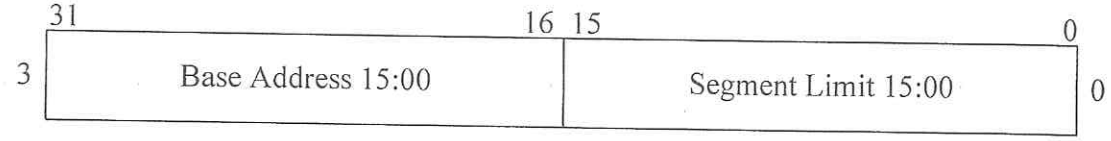
Data-Segment Descriptor:





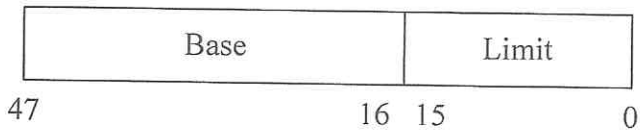
Code-Segment Descriptor:



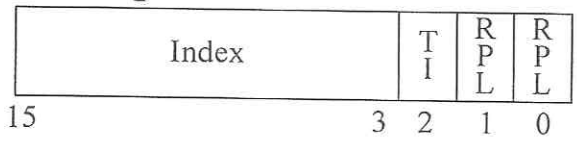


Continued ...

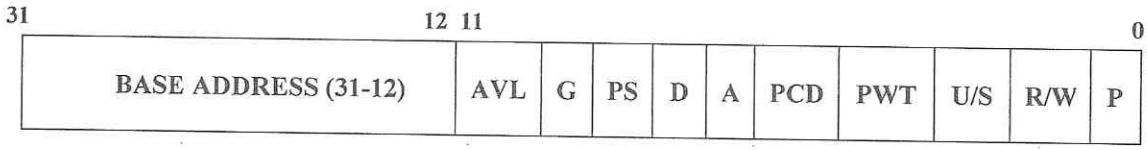
Global Descriptor Table Register (GDTR) and Interrupt Descriptor Table Register (IDTR):



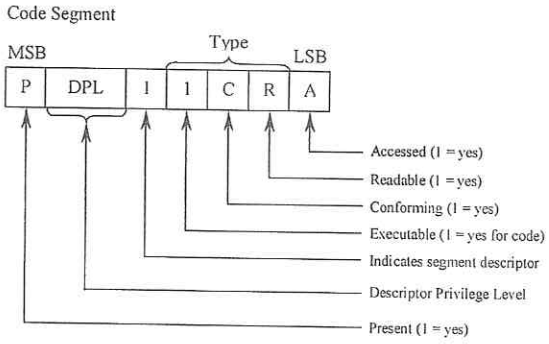
Selector Register:

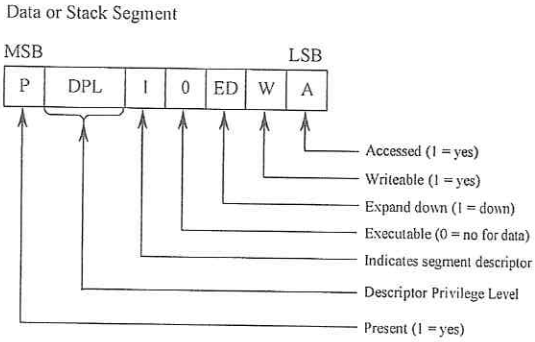


Directory or Page Table Entry Format:



Segment Descriptor:





End of Paper